Administración de datos

Remo Suppi Boldrito

P07/M2103/02287



Índex

Introducció

Un aspecto importante de un sistema operativo es dónde y cómo se guardan los datos. Cuando la disponibilidad de estos datos debe ser eficiente, es necesaria la utilización de bases de datos (DB).

Una base de datos es un conjunto estructurado de datos que pueden ser organizados de manera simple y eficiente por un manejador de dicha base. Las bases de datos actuales se denominan relacionales, ya que los datos pueden ser almacenados en diferentes tablas que facilitan su gestión y administración. Para ello y con el fin de estandarizar el acceso a las bases de datos se utiliza un lenguaje denominado SQL (*structured query language*), que permite una interacción flexible, rápida e independiente de las aplicaciones a las bases de datos.

La forma más utilizada en la actualidad consiste en acceder a una base de datos desde una aplicación que ejecuta código SQL. Por ejemplo, es muy común acceder a una DB a través de una página web que contiene código PHP o Perl (los más comunes). Cuando se solicita una página por parte de un cliente, se ejecuta el código PHP o Perl incrustado en la página, se accede a la DB y se genera la página con su contenido estático y el contenido extraído de la DB que posteriormente se envía al cliente. Dos de los ejemplos más actuales de bases de datos son los aportados por ProstgreSQL y MySQL, que serán objeto de nuestro análisis.

Sin embargo, cuando se trabaja en el desarrollo de un software, existen otros aspectos relacionados con los datos, que son su validez y su ámbito (sobre todo si existe un conjunto de usuarios que trabajan sobre los mismos datos). Existen diversos paquetes para el **control de versiones** (revisiones), pero el objetivo de todos ellos es facilitar la administración de las distintas versiones de cada producto desarrollado junto con las posibles especializaciones realizadas para algún cliente específico.

El control de versiones se realiza para controlar las distintas versiones del código fuente. Sin embargo, los mismos conceptos son aplicables a otros ámbitos y no sólo para código fuente sino para documentos, imágenes, etcétera. Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión (CVS, Subversion, SourceSafe, Clear Case, Darcs, Plastic SCM, RCS, etc.).

En este capítulo veremos CVS (*control version system*) y Subversion para controlar y administrar múltiples revisiones de archivos, automatizando el almacenamiento, lectura, identificación y mezclado de diferentes revisiones. Estos programas son útiles cuando un texto es revisado frecuentemente e incluye

código fuente, ejecutables, bibliotecas, documentación, gráficos, artículos y otros archivos. [Pos03e, Mys, Ced]

La justificación de CVS y Subversion se puede encontrar en que CVS es uno de los paquetes tradicionales y más utilizados y Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS y soluciona varias deficiencias de éste. Al Subversion también se lo conoce como svn por ser ése el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos con versiones no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.

1. PostgreSQL

En el lenguaje de bases de datos (DB), PostgreSQL utiliza un modelo cliente servidor [Posa]. Una sesión de PostgreSQL consiste en una serie de programas que cooperan:

- Un proceso servidor que maneja los archivos de la DB acepta conexiones de los clientes y realiza las acciones solicitadas por éstos sobre la DB. El programa servidor es llamado en PostgreSQL *postmaster*.

- La aplicación del cliente (*frontend*) es la que solicita las operaciones que hay que realizar en la DB y que pueden ser de lo más variadas; por ejemplo: herramientas en modo texto, gráficas, servidores de web, etc.

Generalmente, el cliente y el servidor se encuentran en diferentes *hosts* y se comunican a través de una conexión TCP/IP. El servidor puede aceptar múltiples peticiones de diferentes clientes, activando para cada nueva conexión un proceso que lo atenderá en exclusiva de un modo transparente para el usuario. Hay un conjunto de tareas que pueden ser llevadas a cabo por el usuario o por el administrador, según convenga, y que pasamos a describir a continuación.

1.1. ¿Cómo se debe crear una DB?

La primera acción para verificar si se puede acceder al servidor de DB es crear una base de datos. El servidor PostgreSQL puede manejar muchas DB y es recomendable utilizar una diferente para cada proyecto. Para crear una base de datos, se utiliza el comando createdb desde la línea de comandos del sistema operativo. Este comando generará un mensaje *CREATE DATABASE* si todo es correcto. Es importante tener en cuenta que para esta acción debe haber un usuario habilitado para crear una base de datos. Se verá en el apartado de instalación (8.1.4) que existe un usuario, el que instala la base de datos, que tendrá permisos para crear bases de datos y crear nuevos usuarios que a su vez puedan crear bases de datos. Generalmente (y en Debian) este usuario es postgres por defecto. Por ello, antes de hacer el createdb, se debe hacer un postgres (si se es *root,* no es necesaria ninguna palabra clave, pero si se es otro usuario, necesitaremos la palabra clave de postgres) y luego se podrá realizar el createdb. Para crear una DB llamada nteumdb:

```
createdb nteumdb
```

Si no encontráis el comando, puede ser que no esté bien configurado el camino o que la DB esté mal instalada. Se puede intentar con todo el camino

(/usr/local/pgsql/bin/createdb nteumdb), que dependerá de la instalación que se haya hecho, o consultar referencias para la solución de problemas. Otros mensajes de error serían *could not connect to server* cuando el servidor no está arrancado o *CREATE DATABASE: permission denied* cuando no se tienen privilegios para crear la DB. Para eliminar la base de datos, se puede utilizar `dropdb nteumdb`.

1.2. ¿Cómo se puede acceder a una DB?

Una vez creada la DB, se puede acceder a ella de diversas formas:

- Ejecutando un comando interactivo llamado psql, que permite editar y ejecutar comandos SQL.(p. ej. psql nteumdb)
- Ejecutando una interfaz gráfica como PgAccess o alguna *suite* que tenga soporte ODBC para crear y manipular DB.
- Escribiendo una aplicación utilizando algunos de los lenguajes soportados, por ejemplo, PHP, Perl, Java... (ver PostgreSQL 7.3 Programmer's Guide).

Nota

Para poder acceder a la DB, el servidor de base de datos deberá estar en funcionamiento. Cuando se instala Postgresql se crean los enlaces adecuados para que el servidor se inicie en el arranque del ordenador. Para más detalles, consultad el apartado de instalación (8.1.4).

Por simplicidad utilizaremos psql para acceder a la DB, por lo que se deberá introducir `psql nteumdb`: saldrán unos mensajes con la versión e información y un prompt similar a nteumdb =>. Se pueden ejecutar algunos de los comandos SQL siguientes:

SELECT version();

o también

SELECT current date;

psql también tienen comandos que no son SQL y comienzan por '\' por ejemplo \h (lista todos los comandos disponibles) o \q para terminar.

Ejemplo

```
Acceder a la DB nteumdb:
psql nteumdb [enter]
nteumdb =>
```

1.3. El lenguaje SQL

No es la finalidad de este apartado hacer un tutorial sobre SQL, pero se analizarán unos ejemplos para ver las capacidades de este lenguaje. Son ejemplos que vienen con la distribución de PostgreSQL en el directorio DirectorioInstalacion/src/tutorial, para acceder a ellos, cambiad al directorio de PostgreSQL (cd DirectorioInstalación/*src*/tutorial) y ejecutad `psql -s nteumdb`

y después, dentro \i basics.sql. El parámetro \i lee los comandos del archivo especificado (basic.sql en nuestro caso).

PostgreSQL es una base de datos relacional (*relational database management system*, RDBMS), lo cual significa que maneja los datos almacenados en tablas. Cada tabla tiene un número determinado de filas y de columnas y cada columna tiene un tipo específico de datos. La tablas se agrupan en una DB y un único servidor maneja esta colección de DB (todo el conjunto se denomina agrupación de bases de datos, *database cluster*).

Para crear, por ejemplo, una tabla con psql, ejecutad:

```
CREATE TABLE tiempo (
        ciudad          varchar(80),
        temp_min        int,
        temp_max        int,
        lluvia          real,
        dia             date
);
```

Ejemplo

Crear tabla. Dentro de *psql*:

CREATE TABLE NombreTB (var1 tipo, var2 tipo,...);

El comando termina cuando se pone ';' y se pueden utilizar espacios en blanco y tabs libremente. varchar(80) especifica una estructura de datos que puede almacenar hasta 80 caracteres (en nuestro caso). El point es un tipo específico de PostgreSQL.

Para borrar la tabla:

DROP TABLE *nombre_tabla*;

Para introducir datos, se pueden utilizar dos formas, la primera es poniendo todos los datos de la tabla y la segunda, indicando las variables y los valores que se desean modificar:

INSERT INTO tiempo VALUES ('Barcelona', 16, 37, 0.25, '2007-03-19');
INSERT INTO tiempo (ciudad, temp_min, temp_max, lluvia, dia)

VALUES ('Barcelona', 16, 37, 0.25, '2007-03-19');
Esta forma puede ser sencilla para unos pocos datos, pero cuando hay que introducir gran cantidad de datos, se pueden copiar desde un archivo con la sentencia:

COPY tiempo FROM '/home/user/tiempo.txt'; (este archivo debe estar en el servidor, no en el cliente).

Para mirar una tabla, podríamos hacer:

SELECT * FROM tiempo;

donde el * significa todas las columnas.

Nota

Un segundo ejemplo podría ser:

```
CREATE TABLE ciudad (
nombre varchar(80),
        lugar
        point
);
```

Nota

Se recomienda ver el capítulo 3 de PostgreSQL sobre características avanzadas (Views, Foreign Keys, Transactions, http://www.postgresql.org/docs/8.2/static/tutorial-advanced.html

[Pos03d]

Ejemplos

Introducir datos en tabla. Dentro de psql:

INSERT INTO NombreTB (valorVar1, ValorVar2,...);

Datos desde un archivo. Dentro de psql:

COPY NombreTB FROM 'NombreArchivo';

Visualizar datos. Dentro de *psql*:

SELECT * FROM NombreTB;

Ejemplos de comandos más complejos serían (dentro de *psql*):

- Visualiza la columna ciudad después de realizar la operación:

SELECT ciudad, (temp_max+temp_min)/2 AS temp_media, date FROM tiempo;

- Visualiza todo donde se cumple la operación lógica:

SELECT * FROM tiempo WHERE city = 'Barcelona'
AND lluvia \verb+>+ 0.0;

- Unión de tablas:

SELECT * FROM tiempo, ciudad WHERE ciudad = nombre;

- Funciones, máximo en este caso:

SELECT max(temp_min) FROM tiempo;

- Funciones anidadas:

SELECT ciudad FROM tiempo WHERE temp_min = (SELECT max(temp_min)
FROM tiempo);

- Modificación selectiva:

UPDATE tiempo SET temp_max = temp_max 2, temp_min = temp_min 2
WHERE dia > '19990128';

- Borrado del registro:

DELETE FROM tiempo WHERE ciudad = 'Sabadell';

1.4. Instalación PostgreSQL

Este paso es necesario para los administradores de la DB [Posa]. Dentro de las funciones del administrador de DB se incluye en la instalación del software, inicialización y configuración, administración de los usuarios, DBs y tareas de mantenimiento de la DB.

La instalación de la base de datos se puede realizar de dos modos, a través de los binarios de la distribución, lo cual no presenta ninguna dificultad, ya que los *scripts* de distribución realizan todos los pasos necesarios para tener la DB operativa, o a través del código fuente, que será necesario compilar e instalar. En el primer caso, se puede utilizar (Debian) el *kpackage* o el *apt-get*. Para el segundo caso, se recomienda siempre ir al origen (o a un repositorio espejo de la distribución original). Es importante tener en cuenta que la instalación des-

de el código fuente quedará luego fuera de la DB de software instalado y se perderán los beneficios de administración de software que presente por ejemplo *apt-cache* o *apt-get*.

Instalación desde el código fuente paso a paso

- Primero se debe obtener el software del sitio (x.x es la versión disponible) http://www.postgresql.org/download/ y se descomprime (x.x.x es la versión 8.2.3 al momento de hacer esta revisión):

```
gunzip postgresql-x.x.x.tar.gz
tar xf postgresql-7.3.tar
```

- Cambiarse al directorio postgresql y configurarlo con ./configure.

- Compilarlo con gmake, verificar la compilación con `gmake check` e instalarlo con `gmake install` (por defecto, lo hará en /usr/local/pgsql).

1.4.1. Postinstalación

Inicializar las variables, en *bash*, *sh*, *ksh*:

```
LD_LIBRARY_PATH = /usr/local/pgsql/lib;
PATH = /usr/local/pgsql/bin:$PATH;
export LD_LIBRARY_PATH PATH;
```

o bien, en csh:

```
setenv LD_LIBRARY_PATH /usr/local/pgsql/lib;
set path = (/usr/local/pgsql/bin $path)
```

Es recomendable poner esta inicialización en los *scripts* de configuración del usuario, por ejemplo /etc/profile o .bashrc para el *bash*. Para tener acceso a los manuales, se debe inicializar la variable MANPATH de la misma forma:

```
MANPATH = /usr/local/pgsql/man:$MANPATH;
export MANPATH
```

Una vez instalada la DB, se deberá crear un usuario que manejará las bases de datos (es conveniente crear un usuario diferente del *root* para que no tenga conexión con otros servicios de la máquina), por ejemplo, el usuario postgres utilizando el comando useradd, por ejemplo.

A continuación, se debe crear un área de almacenamiento para las bases de datos (espacio único) sobre el disco que será un directorio, por ejemplo /usr/local/pgsql/data. Para ello, ejecutar el comando `initdb -D /usr/local/pgsql/data`, conectado como el usuario creado en el punto anterior. Puede recibir un mensaje que no puede crear el directorio por falta de privilegios, por lo cual se deberá

crear el directorio primero y luego indicarle a la DB cuál es; como *root,* hay que hacer, por ejemplo:

```
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data su postgres
initdb -D /usr/local/pgsql/data
```

Iniciar el servidor (que se llama *postmaster*), para ello, utilizar:

```
postmaster -D /usr/local/pgsql/data
```

para ejecutarla en modo activo (*foreground*); y para ejecutarlo en modo pasivo (*background*) utilizar:

```
postmaster -D /usr/local/pgsql/data > logfile 2>&1 &.
```

Las redirecciones se hacen para almacenar los errores del servidor. El paquete también incluye un *script* (pg_ctl) para no tener que conocer toda la sintaxis de *postmaster* para ejecutarlo:

```
/usr/local/pgsql/bin/pg_ctl start -l logfile \
-D /usr/local/pgsql/data
```

Para abortar la ejecución del servidor, se puede hacer de diferentes formas, con el pg-ctl, por ejemplo, o bien directamente con:

```
kill -INT 'head -1 /usr/local/pgsql/data/postmaster.pid'
```

1.4.2. Usuarios de DB

Nota

Crear, borrar usuarios:
createuser [opciones] nombre
dropuser [opciones

Los usuarios de la DB son completamente distintos de los usuarios del sistema operativo. En algunos casos podría ser interesante mantener una correspondencia, pero no es necesario. Los usuarios son para todas las DB que controla dicho servidor, no para cada DB. Para crear un usuario, ejecutad la sentencia SQL:

CREATE USER *nombre*

Para borrar usuarios:

DROP USER *nombre*

También se puede llamar a los programas createuser y dropuser desde la línea de comandos. Existe un usuario por defecto llamado *postgres* (dentro de la DB), que es el que permitirá crear los restantes (para crear nuevos usuarios desde `psql -U postgres` si el usuario de sistema operativo con el que se administra la DB no es *postgres*).

Un usuario de DB puede tener un conjunto de atributos en función de lo que puede hacer:

- Superusuario: este usuario no tiene ninguna restricción. Por ejemplo, podrá crear nuevos usuarios; para ello, ejecutar:

  CREATE USER nombre CREATEUSER

- Creador de DB: tiene permiso para crear DB. Para crear un usuario de estas características, utilizar el comando:

  CREATE USER nombre CREATEDB

- Password: sólo es necesario si por cuestiones de seguridad se desea controlar el acceso de los usuarios cuando se conecten a una DB. Para crear un usuario con contraseña, se puede utilizar:

  CREATE USER nombre PASSWORD 'palabra_clave'

  donde palabra_clave será la clave para ese usuario.

- A un usuario se le pueden cambiar los atributos utilizando el comando ALTER USER. También se pueden hacer grupos de usuarios que compartan los mismos privilegios con:

  CREATE GROUP NomGrupo

  Y para insertar usuarios en este grupo:

  ALTER GROUP NomGrupo ADD USER Nombre1

  O para borrar:

  ALTER GROUP NomGrupo DROP USER Nombre1

  Ejemplo

  Operaciones con grupo dentro de psql:

  CREATE GROUP *NomGrupo*;
  ALTER GROUP *NomGrupo* ADD USER *Nom1*...; ALTER GROUP *NomGrupo*
  DROP USER *Nom1*...;

Cuando se crea una DB, los privilegios son para el usuario que la crea (y para el *superuser*). Para permitir que otro usuario utilice esta DB o parte de ella, se le deben conceder privilegios. Hay diferentes tipos de privilegios como SELECT, INSERT, UPDATE, DELETE, RULE, REFERENCES, TRIGGER, CREATE, TEMPORARY, EXECUTE, USAGE, y ALL PRIVILEGES (consultad las referencias para ver su significado). Para asignar los privilegios, se puede utilizar:

GRANT UPDATE ON objeto TO usuario

donde usuario deberá ser un usuario válido de PostgerSQL y objeto, una tabla, por ejemplo. Este comando lo deberá ejecutar el superusuario o el dueño de la

tabla. El usuario PUBLIC puede ser utilizado como sinónimo de todos los usuarios y ALL, como sinónimo de todos los privilegios. Por ejemplo, para quitar todos los privilegios a todos los usuarios de objeto, se puede ejecutar:

REVOKE ALL ON objeto FROM PUBLIC;

1.5. Mantenimiento

Hay un conjunto de tareas que son responsabilidad del administrador de DB y que se deben realizar periódicamente:

1) Recuperar el espacio: se deberá ejecutar periódicamente el comando VACUUM, el cual recuperará el espacio de disco de filas borradas o modificadas, actualizará las estadísticas utilizadas por el planificador de PostgreSQL y mejorará las condiciones de acceso.

2) Reindexar: PostgreSQL en ciertos casos puede dar algunos problemas con la reutilización de los índices, por ello es conveniente utilizar REINDEX periódicamente para eliminar páginas y filas. También se puede utilizar `contrib/reindexdb` para reindexar una DB entera (se debe tener en cuenta que, dependiendo del tamaño de las DB, estos comandos pueden tardar un cierto tiempo).

3) Cambio de archivos *log*: se debe evitar que los archivos de *log* sean de tamaño muy grande y difíciles de manejar. Se puede hacer fácilmente cuando se inicia el servidor con:

```
pg_ctl start | logrotate
```

logrotate renombra y abre un nuevo archivo de *log* y se puede configurar con /etc/logrotate.conf.

4) Copia de seguridad y recuperación (*backup* y *recovery*): existen dos formas de salvar los datos, por la sentencia SQL Dump o salvando el archivo de la DB. El primero será:

```
pg_dump ArchivoDB > ArchivoBackup
```

Para recuperar, se puede utilizar: `psql ArchivoDB < ArchivoBackup`

Para salvar todas las DB del servidor, se puede ejecutar:

```
pg_dumpall > ArchivoBackupTotal
```

Otra estrategia es salvar los archivos de las bases de datos a nivel del sistema operativo, por ejemplo con:

```
tar -cf backup.tar /usr/local/pgsql/data
```

Existen dos restricciones que pueden hacer este método poco práctico:

- El servidor debe ser parado antes de salvar y de recuperar los datos.
- Deben conocerse muy bien todas las implicaciones a nivel archivo donde están todas las tablas, transacciones y demás, ya que si no, una DB puede quedar inútil. Además (generalmente), el tamaño que se salvará será mayor que el efectuado con los métodos anteriores, ya que por ejemplo, con el pg_dumb no se salvan los índices, sino el comando para recrearlos.

Resumen de la Instalación de PostgreSQL:

```
./configure
gmake
su
gmake install
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data >logfile 2>&1 &
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test
```

1.6. Pgaccess

La aplicación pgaccess [NombreDB] (http://www.pgaccess.org/) permite acceder y administrar una base de datos con una interfaz gráfica. La forma más fácil de acceder (por ejemplo, desde KDE) es desde una terminal, el administrador de DB deberá hacer, (si no es el usuario *postgresql*) `xhost+` lo cual permite que otras aplicaciones puedan conectarse al display del usuario actual

```
su postgres pgaccess [NombreDB]&
```

Si se configura en 'Preferencias' abrirá siempre la última DB. La figura 15 muestra la interfaz de pgaccess.

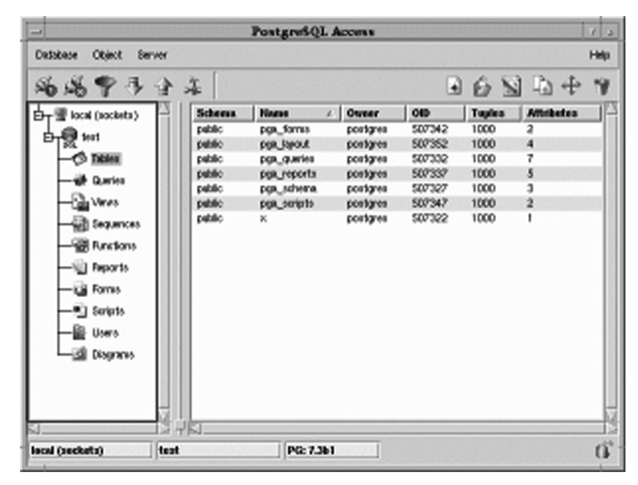


Figura 1. PgAccess

En una sesión típica el administrador/usuario podría, en primer lugar, Abrir Base de Datos, indicando aquí, por ejemplo, Port = 5432, Base de Datos = nteum (los otros parámetros no son necesarios si la base de datos es local) y luego Abrir. A partir de este momento, el usuario podrá trabajar con el espacio bidimensional seleccionando qué es lo que desea hacer en el eje *Y* (tablas, consultas, vistas, etc.) y con ese elemento seleccionado, y escogiendo uno de ese tipo dentro de la ventana, utilizar el eje *X* superior para Nuevo (añadir), Abrir o Diseño. Por ejemplo, si se selecciona en *Y* Usuarios y en *X,* Nuevo, la aplicación solicitará el nombre del usuario, su contraseña (con verificación), su validez en el tiempo y sus características (por ejemplo, Crear DB, Crear otros usuarios). En Base de Datos también se podría seleccionar Preferencias para, por ejemplo, cambiar el tipo de fuente y seleccionar la posibilidad de ver las tablas del sistema.

Las configuraciones personales de los usuarios quedarán registradas en el archivo ~/.pgaccessrc. La interfaz permite realizar/agilizar gran parte del trabajo del usuario/administrador y es recomendable para usuarios que se acaban de iniciar en PostgreSQL, ya que no necesitan conocer la sintaxis de la línea de comando como en psql (la propia aplicación solicitará mediante múltiples ventanas todas las opciones de un comando).

Una herramienta más simple es a través del módulo correspondiente de webmin (es necesario instalar los paquetes webmin-core y los módulos necesarios, por ejemplo en este caso webmin-postgresql), pero su inclusión en muchas distribuciones no está por defecto (consultar más información en http://www.webmin.com/). Durante la instalación, el webmin dará un aviso de que el usuario principal será el *root* y utilizará la misma contraseña que el *root* del sistema operativo. Para conectarse, se podrá hacer, por ejemplo, desde un navegador, https://localhost:10000, el cual solicitará aceptar (o denegar) la utilización del certificado para la comunicación SSL y, a continuación, mostrará todos los servicios que puede administrar, entre ellos PostgreSQL Data Base Server.

2. Mysql

MySQL [Mys] es (según sus autores) la base de datos (DB) SQL abierta, es decir, software libre (Open Source) más popular, y es desarrollada y distribuida por MySQL AB (compañía comercial que obtiene sus beneficios de los servicios que provee sobre la DB). MySQL es un DBMS (*database management system*). Un DBMS es el que puede añadir y procesar los datos almacenados dentro de la DB. Al igual que PostgreSQL, MySQL es una base de datos relacional, lo que significa que almacena los datos en tablas en lugar de una única ubicación, lo cual permite mayor velocidad y flexibilidad. Al ser software libre, cualquiera puede obtener el código, estudiarlo y modificarlo de acuerdo a sus necesidades sin pago alguno, ya que MySQL utiliza licencia GPL. MySQL provee en su página web un conjunto de estadísticas y prestaciones en comparación con otras DB para mostrar al usuario cuán rápida, fiable y fácil es de usar. La decisión de elegir una DB se debe hacer cuidadosamente en función de las necesidades de los usuarios y del entorno donde se utilizará esta DB.

2.1. Instalación

- Obtener desde http://www.mysql.com/ o desde cualquiera de los repositorios de software. Se pueden obtener los binarios y los archivos fuente para compilarlos e instalarlos.

- En el caso de los binarios, utilizar la distribución de Debian y seleccionar los paquetes mysql-* (client, server, common son necesarios). La instalación, después de unas preguntas, creará un usuario mysql y una entrada en /etc/init.d/mysql para arrancar/parar el servidor en el *boot*. También se puede hacer manualmente haciendo:

  ```
  /etc/init.d/mysql start|stop
  ```

- Para acceder a la base de datos, se puede utilizar el monitor mysql desde la línea de comando. Si obtiene los binarios (no Debian ni RPM, con esto simplemente utilizar las comunes -apt-get, rpm-), por ejemplo gz desde el sitio web de MySQL, deberá ejecutar los siguientes comandos para instalar la DB:

```
groupadd mysql
useradd -g mysql mysql
cd /usr/local
gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
ln -s full-path-to-mysql-VERSION-OS mysql
cd mysql
scripts/mysql_install_db --user=mysql
chown -R root .
chown -R mysql data
chgrp -R mysql .
bin/mysqld_safe --user=mysql &
```

Esto crea el usuario/grupo/directorio, descomprime, e instala la DB en /usr/local/mysql.

- En caso de obtener el código fuente, los pasos son similares:

```
groupadd mysql
useradd -g mysql mysql
gunzip < mysql-VERSION.tar.gz | tar -xvf -
cd mysql-VERSION
./configure --prefix=/usr/local/mysql
make
make install
cp support-files/my-medium.cnf /etc/my.cnf
cd /usr/local/mysql
bin/mysql_install_db --user=mysql
chown -R root .
chown -R mysql var
chgrp -R mysql .
bin/mysqld_safe --user=mysql &
```

Es importante prestar atención cuando se realiza el configure, ya que `prefix= /usr/local/mysql` es el directorio donde se instalará la DB y se puede cambiar para ubicar la DB en el directorio que se desee.

2.2. Postinstalación y verificación

Una vez instalada (ya sea de los binarios o del código fuente), se deberá verificar si el servidor funciona correctamente. En Debian se puede hacer directamente:

`/etc/init.d/mysql start`	Inicia el servidor
`mysqladmin version`	Genera información de versiones
`mysqladmin variables`	Muestra los valores de las variables
`mysqladmin -u root shutdown`	Finaliza ejecución del servidor
`mysqlshow`	Mostrará las DB predefinidas
`mysqlshow mysql`	Mostrará las tablas de la DB mysql

Si se instala desde el código fuente, antes de hacer estas comprobaciones se deben ejecutar los siguientes comandos para crear las bases de datos (desde el directorio de la distribución):

```
./scripts/mysql_install_db
cd DirectorioInstalacionMysql
./bin/mysqld_safe --user = mysql &
```

Si se instala de binarios (RPM, Pkg,...), se debe hacer lo siguiente:

```
cd DirectorioInstalacionMysql
./scripts/mysql_install_db
./bin/mysqld_safe user = mysql &
```

El script `mysql_install_db` crea la DB `mysql` y `mysqld_safe` arranca el servidor mysqld. A continuación, se pueden probar todos los comandos dados anteriormente para Debian, excepto el primero que es el que arranca el servidor. Además, si se han instalado los tests, se podrán ejecutar con `cd sql-bench` y luego `run-all-tests`. Los resultados se encontrarán en el directorio sql-bech/Results para compararlos con otras DB.

2.3. El programa monitor (cliente) mysql

Nota

Cliente (frontend) mysql:
mysql [NombreDB]

Nota

Para más información, podéis consultar la documentación, comandos y opciones. [Mys07]

http://dev.mysql.com/doc/refman/5.0/es/

El cliente mysql se puede utilizar para crear y utilizar DB simples, es interactivo y permite conectarse al servidor, ejecutar búsquedas y visualizar los resultados. También funciona en modo *batch* (como un *script*) donde los comandos se le pasan a través de un archivo. Para ver todas las opciones del comando, se puede ejecutar mysql --help. Podremos realizar una conexión (local o remota) con el comando mysql, por ejemplo, para una conexión por la interfaz de red pero desde la misma máquina:

```
mysql -h localhost -u mysql -p NombreDB
```

Si no se pone el último parámetro, ninguna DB es seleccionada.

Una vez dentro, el mysql pondrá un prompt (mysql>) y esperará a que le introduzcamos algún comando (propio y SQL), por ejemplo help. A continuación, daremos una serie de comandos para probar el servidor (recordar poner siempre el ';' para terminar el comando):

```
mysql> SELECT VERSION(), CURRENT_DATE;
```

Se puede utilizar mayúsculas o minúsculas.

```
mysql> SELECT SIN(PI()/4), (4+1)*5; Calculadora.
mysql> SELECT VERSION(); SELECT NOW();
```

Múltiples comandos en la misma línea.

```
mysql> SELECT
-> USER()
-> ,
-> CURRENT_DATE;
```

O en múltiples líneas.

```
mysql> SHOW DATABASES;
```

Muestra las DB disponibles.

```
mysql> USE test
```

Cambia la DB.

```
mysql> CREATE DATABASE nteum; USE nteum;
```

Crea y selecciona una DB llamada nteum.

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
-> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

Crea una tabla dentro de nteum.

```
mysql> SHOW TABLES;
```

Muestra las tablas.

```
mysql> DESCRIBE pet;
```

Muestra la definición de la tabla.

```
mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```

Carga datos desde pet.txt en pet. El archivo pet.txt debe tener un registro por línea separado por tabs de los datos de acuerdo a la definición de la tabla (fecha en formato AAAA-MM-DD)

```
mysql> INSERT INTO pet
-> VALUES ('Marciano','Estela','gato','f','1999-03-30',NULL);
```

Carga los datos in-line.

```
mysql> SELECT * FROM pet;Muestra los datos de la tabla.
mysql> UPDATE pet SET birth = "1989-08-31" WHERE name = "Browser";
```

Modifica los datos de la tabla.

```
mysql> SELECT * FROM pet WHERE name = "Browser";
```

Muestra selectiva.

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
```

Muestra ordenada.

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```

Muestra selectiva con funciones.

```
mysql> GRANT ALL PRIVILEGES ON *.* TO marciano@localhost ->
IDENTIFIED BY 'passwd' WITH GRANT OPTION;
```

Crea un usuario marciano en la DB. Lo debe hacer el *root* de la DB. O también se puede hacer directamente con:

```
mysql> INSERT INTO user (Host,User,Password) ->
VALUES('localhost','marciano','passwd');
```

2.4. Administración

Mysql dispone de un archivo de configuración en /etc/mysql/my.cnf (en Debian), al cual se le pueden cambiar las opciones por defecto a la DB, como por

ejemplo, el puerto de conexión, usuario, contraseña de los usuarios remotos, archivos de log, archivos de datos, si acepta conexiones externas, etc. Con respecto a la seguridad, se deben tomar algunas precauciones:

1) No dar a nadie (excepto al usuario *root* de mysql) acceso a la tabla user dentro de la DB mysql, ya que aquí se encuentran las contraseñas de los usuarios que podrían ser utilizados con otros fines.

2) Verificar `mysql -u root`. Si se puede acceder, significa que el usuario *root* no tiene contraseña. Para cambiarlo, se puede hacer:

```
mysql -u root mysql
mysql> UPDATE user SET Password = PASSWORD('new_password')
-> WHERE user = 'root';
mysql> FLUSH PRIVILEGES;
```

Ahora, para conectarse como *root*:

```
mysql -u root -p mysql
```

3) Comprobar la documentación (punto 4.2) respecto a las condiciones de seguridad y del entorno de red para evitar problemas de ataques y/o intrusión.

4) Para hacer copias de la base de datos, se puede utilizar el comando:

```
mysqldump --tab = /DirectorioDestino --opt NombreDB
```

o también:

```
mysqlhotcopy NombreDB /DirectorioDestino
```

Asimismo, se pueden copiar los archivos *.frm', *.MYD', y *.MYI con el servidor parado. Para recuperar, ejecutad:

REPAIR TABLE o myisamchk –r

lo cual funcionará en el 99% de las veces. En caso contrario, podría copiar los archivos salvados y arrancar el servidor. Existen otros métodos alternativos en función de lo que se quiera recuperar, como la posibilidad de salvar/recuperar parte de la DB (consultar punto 4.4 de la documentación). [Mys]

2.5.Interfaces gráficas

Para Mysql hay gran cantidad de interfaces gráficas, entre las que destacamos Mysql Administrator (se puede obtener desde http://www.mysql.com/products/tools/administrator/). También pueden servir como herramientas Mysql-Navigator (http://sourceforge.net/projects/mysqlnavigator/), o Webmin con el módulo para trabajar con Mysql (paquetes webmin-core y webmin-mysql) si bien este último ya no se incluye con algunas distribuciones. En forma

análoga a PostgreSQL, webmin permite también trabajar con Mysql (es necesario instalar los paquetes webmin-mysql además del webmin-core). Durante la instalación, el webmin dará un aviso de que el usuario principal será el root y utilizará la misma contraseña que el *root* del sistema operativo. Para conectarse, se podrá hacer, por ejemplo, desde un navegador: https://localhost:10000, el cual solicitará aceptar (o denegar) la utilización del certificado para la comunicación SSL y a continuación mostrará todos los servicios que puede administrar, entre ellos Mysql Data Base Server.

MySQL Administrator es una aplicación potente para la administración y control de bases de datos basadas en MySQL. Esta aplicación integra la gestión y control de la BD y el mantenimiento en forma simple y en un mismo entorno. Las características principales son: administración avanzada de grandes DB, reducción de errores a través de una “administración visual”, mayor productividad y un entorno seguro de gestión. La figura siguiente muestra un aspecto de MySQL Administrator (en http://dev.mysql.com/doc/administrator/en/index.html se puede encontrar toda la documentación para su instalación y puesta en marcha).

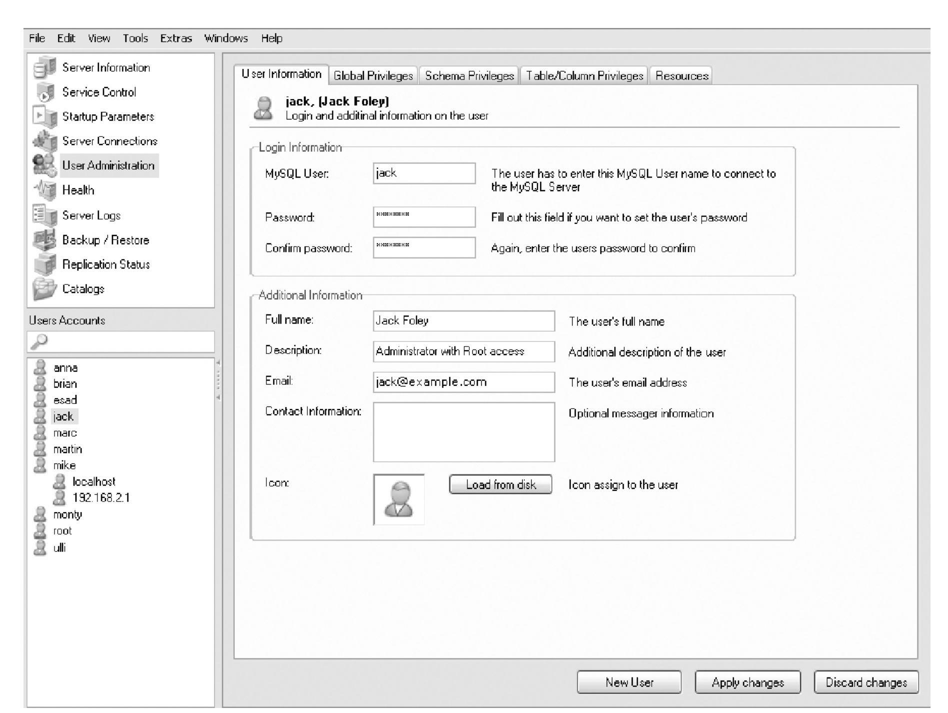


Figura 2. MySQL Administrator

3. ***Source code control system*** **(CVS y Subversion)**

El *concurrent versions system* (CVS) es un sistema de control de versiones que permite mantener versiones antiguas de archivos (generalmente código fuente), guardando un registro (log) de quién, cuándo y por qué fueron realizados los cambios. A diferencia de otros sistemas, CVS no trabaja con un archivo/directorio por vez, sino que actúa sobre colecciones jerárquicas de los directorios que controla.

El CVS tiene por objetivo ayudar a manejar versiones de software y controla la edición concurrente de archivos fuente por múltiples autores. El CVS utiliza otro paquete llamado RCS (*revision control system*) internamente como una capa de bajo nivel. Si bien el RCS puede ser utilizado independientemente, no se aconseja, ya que CVS además de su propia funcionalidad presenta todas las prestaciones de RCS pero con notables mejoras en cuanto a la estabilidad, operación y mantenimiento. Entre ellas podemos destacar: funcionamiento descentralizado (cada usuario puede tener su propio árbol de código), edición concurrente, comportamiento adaptable mediante *shell scripts*, etc. [Ced, CVS, Vasa, Kie]

Como ya se ha explicado en la introducción Subversion (http://subversion.tigris.org/) es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS, y extender sus capacidades. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como `svn` por el nombre en línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente y en su lugar todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en el tiempo que se ha "versionado". Entre las características principales podemos mencionar:

- Se sigue la historia de los archivos y directorios a través de copias y renombrados.
- Las modificaciones atómicas y seguras (incluyendo cambios a varios archivos).
- El creado de ramas y etiquetas es eficiente y simple.
- Se envían sólo las diferencias en ambas direcciones (en CVS siempre se envían al servidor archivos completos).
- Puede ser servido, mediante Apache, sobre WebDAV/DeltaV.

- Maneja eficientemente archivos binarios (a diferencia de CVS que los trata internamente como si fueran de texto).

Existe un libro (libre) interesante que explica todo lo referente al Subversion http://svnbook.red-bean.com/index.es.html y la traducción está bastante avanzada (http://svnbook.red-bean.com/nightly/es/index.html).

3.1. *Revision control system* (RCS)

Ya que CVS se basa en RCS y en algunos sistemas todavía se utiliza, se darán unas breves explicaciones al respecto. El RCS está formado por un conjunto de programas para las diferentes actividades del RCS: rcs (programa que controla los atributos de los archivos bajo RCS), ci y co (verifican la entrada y la salida de los archivos bajo el control de RCS), ident (busca en el RCS los archivos por palabras claves/atributos), rcsclean (limpia archivos no utilizados o que no han cambiado), rcsdiff (ejecuta el comando diff para comparar versiones), rcsmerge (une dos ramas (archivos) en un único archivo), y rlog (imprime los mensajes de log).

El formato de los archivos almacenados por RCS puede ser texto u otro formato, como por ejemplo binario. Un archivo RCS consiste en un archivo de revisión inicial llamado 1.1 y una serie de archivos de cambios, uno por cada revisión. Cada vez que se realiza una copia del repositorio hacia el directorio de trabajo con el comando co (obtiene una revisión de cada archivo RCS y lo pone en el archivo de trabajo) o se utiliza ci (almacena nuevas revisiones en el RCS), el número de versión se incrementa (por ejemplo, 1.2, 1.3,...). Los archivos (generalmente) están en el directorio /RCS y es necesario que el sistema operativo tenga instalados los comandos diff y diff3 para que funcione adecuadamente. En Debian no es necesario compilarlo ya que se incluye en la distribución.

Con el comando rcs crearemos y modificaremos los atributos de los archivos (consultar man rcs). La forma más fácil de crear un repositorio es hacer en primer lugar un `mkdir rcs` en el directorio de originales e incluir los originales en el repositorio con: `ci nombre_archivos_fuentes`.

Se puede utilizar el * y siempre tener una copia de resguardo para evitar problemas. Esto creará las versiones de los archivos con nombre `./RCS/nombre_archivo` y solicitará un texto para describir el archivo. Luego, con `co RCS/nombre_archivo`, obtendremos una copia de trabajo desde el repositorio. Se puede bloquear o desbloquear este archivo para evitar modificaciones, respectivamente, con:

```
rcs -L nombre_archivo_de_trabajo
rcs -U nombre_archivo_de_trabajo
```

Con rlog nombre_del_archivo podremos ver la información sobre las diferentes versiones. [Kie]

3.2. *Concurrent versions system* (CVS)

En primer lugar se debe instalar el *concurrent versions system* (CVS) desde la distribución teniendo en cuenta que debemos tener instalado RCS y que deberemos instalar también OpenSSH si se le quiere utilizar conjuntamente con CVS para acceso remoto. Las variables de entorno EDITOR CVSROOT deben estar inicializadas por ejemplo en /etc/profile (o en .bash profile):

export EDITOR = /bin/vi
export CVSROOT = /usr/local/cvsroot

Obviamente, los usuarios pueden modificar estas definiciones utilizando /.bash profile. Se debe crear el directorio donde estará el repositorio y configurar los permisos; como *root,* hay que hacer, por ejemplo:

```
export CVSROOT = /usr/local/cvsroot
groupadd cvs
useradd -g cvs -d $CVSROOT cvs
mkdir $CVSROOT
chgrp -R cvs $CVSROOT
chmod o-rwx $CVSROOT
chmod ug+rwx $CVSROOT
```

Para inicializar el repositorio y poner archivo de código en él:

```
cvs -d /usr/local/cvsroot init
```

cvs init tendrá en cuenta no sobrescribir nunca un repositorio ya creado para evitar pérdidas de otros repositorios. Luego, se deberá agregar los usuarios que trabajarán con el CVS al grupo cvs; por ejemplo, para agregar el usuario *nteum*:

```
usermod -G cvs,nteum
```

Ahora el usuario *nteum* deberá introducir sus archivos en el directorio del repositorio (/usr/local/cvsroot en nuestro caso) hacer:

```
export EDITOR = /bin/vi
export CVSROOT = /usr/local/cvsroot
export CVSREAD = yes
cd directorio_de_originales
cvs import NombreDelRepositorio vendor_1_0 rev_1_0
```

El nombre del repositorio puede ser un identificador único o también usuario/proyecto/xxxx si es que el usuario desea tener organizados sus repositorios. Esto creará un árbol de directorios en CVSROOT con esa estructura.

Esto añade un directorio (/usr/local/cvsroot/NombreDelRepositorio) en el repositorio con los archivos que a partir de este momento estarán en el repositorio. Una prueba para saber si se ha almacenado todo correctamente es almacenar una copia en el repositorio y luego crear una copia desde allí y comprobar las diferencias. Por ejemplo, sean los originales en el directorio_del_usuario/dir_org y se desea crear un repositorio como primer_cvs/proj, se deberán ejecutar los siguientes comandos:

`cd dir_org` Cambiar al directorio del código fuente original.

`cvs import -m` “Fuentes originales”

`\primer_cvs/proj usuarioX vers0`
Crea el repositorio en primer_cvs/proj con ususarioX y vers0.

`cd..` Cambiar al directorio superior de dir_org.

`cvs checkout primer_cvs/proj`
Generar una copia del repositorio. La variableCVSROOT debe estar inicializada, si no, se deberá indicar todo el path.

`diff -r dir_org primer_cvs/proj`
Muestra las diferencias entre uno y otro; que no debe haber ninguna excepto por el directorio primer_cvs/proj/CVS que ha creado el CVS.

`rm -r dir_org`
Borra los originales (realizar una copia de resguardo siempre por seguridad y para tener una referencia de dónde se inició el trabajo con el CVS).

La figura siguiente muestra la organización y cómo se distribuyen los archivos entre versiones y ramas.

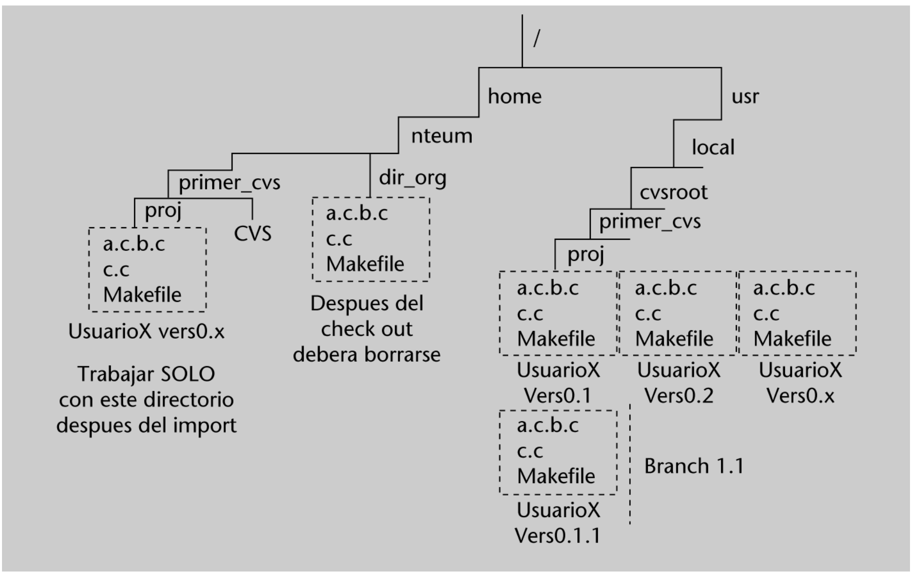


Figura 3

El hecho de borrar los originales no es siempre una buena idea; sólo en este caso, después de que se haya verificado que están en repositorio, para que por descuido no se trabaje sobre ellos y los cambios no queden reflejados sobre el CVS. Sobre máquinas donde los usuarios quieren acceder (por ssh) a un servidor CVS remoto, se deberá hacer:

```
export CVSROOT = ":ext:user@CVS.server.com:/home/cvsroot"
export CVS_RSH = "ssh"
```

Donde user es el login del usuario y cvs.server.com el nombre del servidor donde está CVS. CVS ofrece una serie de comandos (se llaman con cvs comd opciones...) para trabajar con el sistema de revisiones, entre ellos: checkout, update, add, remove, commit y diff.

El comando inicial `cvs checkout` crea su copia privada del código fuente para luego trabajar con ella sin interferir en el trabajo de otros usuarios (como mínimo se crea un subdirectorio donde estarán los archivos).

- `cvs update` se debe ejecutar del árbol privado cuando hay que actualizar sus copias de archivos fuentes con los cambios que otros programadores han hecho sobre los archivos del repositorio.

- `cvs add file...` es un comando necesario cuando hay que agregar nuevos archivos en su directorio de trabajo sobre un módulo donde ya se ha hecho un checkout previamente. Estos archivos serán enviados al repositorio CVS cuando se ejecute el comando cvs commit.

- `cvs import` se puede usar para introducir archivos nuevos en el repositorio.

- `cvs remove file...` este comando se utilizará para borrar archivos del repositorio (una vez que se hayan borrado éstos del archivo privado). Este comando debe ir acompañado de un cvs commit para que los cambios sean efectivos, ya que se trata del comando que transforma todas las peticiones de los usuarios sobre el repositorio.

- `cvs diff file...` se puede utilizar sin que afecte a ninguno de los archivos implicados si se necesita verificar las diferencias entre repositorio y directorio de trabajo o entre dos versiones.

- `cvs tag -R "versión"` se puede utilizar para introducir un número de versión en los archivos de un proyecto y luego hacer un cvs commit y un cvs checkout -r 'version' proyecto para registrar una nueva versión.

Una característica interesante del CVS es poder aislar cambios de los archivos aislados en una línea de trabajo separada llamada *ramificación* (branch). Cuando se cambia un archivo sobre una rama, estos cambios no aparecen sobre los

archivos principales o sobre otras ramas. Más tarde, estos cambios se pueden incorporar a otras ramas o al archivo principal (merging). Para crear una nueva rama, utilizar `cvs tag -b rel-1-0-patches` dentro del directorio de trabajo, lo cual asignará a la rama el nombre de rel-1-0-patches. La unión de ramas con el directorio de trabajo significa utilizar el comando `cvs update -j`. Consultar las referencias para mezclar o acceder a diferentes ramas.

3.2.1. Ejemplo de una sesión

Siguiendo el ejemplo de la documentación dada en las referencias, se mostrará una sesión de trabajo (en forma general) con CVS. Como CVS almacena todos los archivos en un repositorio centralizado, se asumirá que el mismo ya ha sido inicializado anteriormente.

Consideremos que se está trabajando con un conjunto de archivos en C y un *makefile*, por ejemplo. El compilador utilizado es gcc y el repositorio es inicializado a gccrep.

En primer lugar, se debe obtener una copia de los archivos del repositorio a nuestra copia privada con:

```
cvs checkout gccrep
```

que creará un nuevo directorio llamado gccrep con los archivos fuente. Si se ejecuta `cd gccrep` y `ls`, se verá por ejemplo CVS makefile a.c b.c c.c, donde existe un directorio CVS que se crea para el control de la copia privada que normalmente no es necesario tocar.

Después de esto se podría utilizar un editor para modificar a.c e introducir cambios sustanciales en el archivo (ver en la documentación sobre múltiples usuarios concurrentes si se necesita trabajar con más de un usuario en el mismo archivo), compilar, volver a cambiar, etc.

Cuando se decide que se tiene una versión nueva con todos los cambios introducidos en a.c (o en los archivos que sea necesario), es momento de hacer una nueva versión almacenando a.c (o todos los que se han tocado) en el repositorio y hacer esta versión disponible al resto de usuarios: `cvs commit a.c`.

Utilizando el editor definido en la variable CVSEDITOR (o EDITOR si ésta no está inicializada) se podrá introducir un comentario que indique qué cambios se han hecho para que sirva de ayuda a otros usuarios o para recordar qué es lo que caracterizó a esta versión y luego poder hacer un histórico.

Si se decide eliminar los archivos (porque ya se terminó con el proyecto o porque no se trabajará más con él), una forma de hacerlo es a nivel de sistema operativo (rm -r gccrep), pero es mejor utilizar el propio cvs fuera del directorio de trabajo (nivel inmediato superior): `cvs release -d gccrep`. El comando detectará si hay algún archivo que no ha sido enviado al repositorio, y si lo hay y se borra, significa que se perderán todos los cambios, por ello preguntará si se desea continuar o no.

Para mirar las diferencias, por ejemplo, se ha modificado b.c y no se recuerda qué cambios se hicieron, se puede utilizar dentro del directorio de trabajo: `cvs diff b.c`. Éste utilizará el comando del sistema operativo diff para comparar la versión b.c con la versión que se tiene en el repositorio (siempre hay que recordar hacer un `cvs commit b.c` si se desea que estas diferencias sean transferidas al repositorio como una nueva versión).

3.2.2. Múltiples usuarios

Cuando más de una persona trabaja en un proyecto software con diferentes revisiones, es sumamente complicado porque habrá ocasiones en las que más de un usuario se quiera editar el mismo fichero simultáneamente. Una posible solución es bloquear el fichero o utilizar puntos de verificación reservados (*reserved checkouts*), lo cual sólo permitirá a un usuario editar el mismo fichero simultáneamente. Para ello, se deberá ejecutar el comando `cvs admin -l command` (ver man para las opciones).

CVS utiliza un modelo por defecto de puntos no reservados (*unreser- ved checkouts*), que permite a los usuarios editar simultáneamente un fichero de su directorio de trabajo. El primero de ellos que transfiera sus cambios al repositorio lo podrá hacer sin problemas, pero los restantes recibirán un mensaje de error cuando desean realizar la misma tarea, por lo cual, deberán utilizar comandos de cvs para transferir en primer lugar los cambios al directorio de trabajo desde el repositorio y luego actualizar el repositorio con sus propios cambios.

Consultad las referencias para ver un ejemplo de aplicación y otras formas de trabajo concurrente con comunicación entre usuarios. [Vasa].

3.3. Interfaces gráficas

Contamos con un conjunto de interfaces gráficas como tkcvs (http://www.twobarleycorns.net/tkcvs.html) [gcus] desarrollada en Tcl/Tk y que soporta subversión, u otra también muy popular, cervisia [Cerc].

En la wiki de CVS (http://ximbiot.com/cvs/wiki/index.php?title=CVS_Clients) se pueden encontrar también un conjunto de clientes, *plugins* para CVS. A continuación, se muestran dos de la interfaces gráficas mencionadas (tkcvs y gcvs):

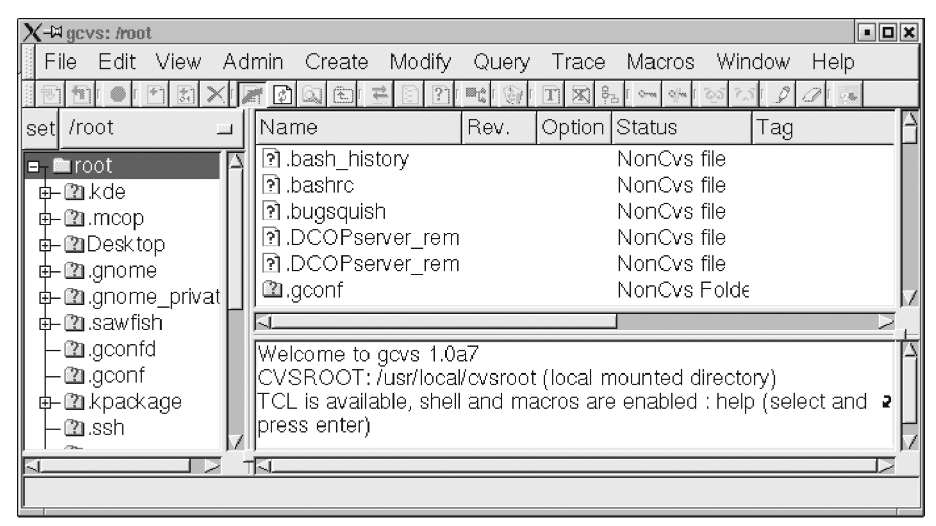


Figura 4. TkCVS (TkSVN)

4. Subversion

Como idea inicial subversion sirve para gestionar un conjunto de archivos (repositorio) y sus distintas versiones. Es interesante remarcar que no nos importa cómo se guardan, sino cómo se accede a estos ficheros, para lo que es común utilizar una base de datos. La idea de repositorio es como un directorio del cual se quiere recuperar un fichero de hace una semana o 10 meses a partir del estado de la base de datos, recuperar las últimas versiones y agregar una nueva. A diferencia de CVS, subversion hace las revisiones globales del repositorio, lo cual significa que un cambio en el fichero no genera un salto de versión únicamente en ese fichero, sino en todo el repositorio, que suma uno a la revisión. Además del libro mencionado (http://svnbook.red-bean.com/nightly/es/index.html) consultar la documentación en http://subversion.tigris.org/servlets/ProjectDocumentList.

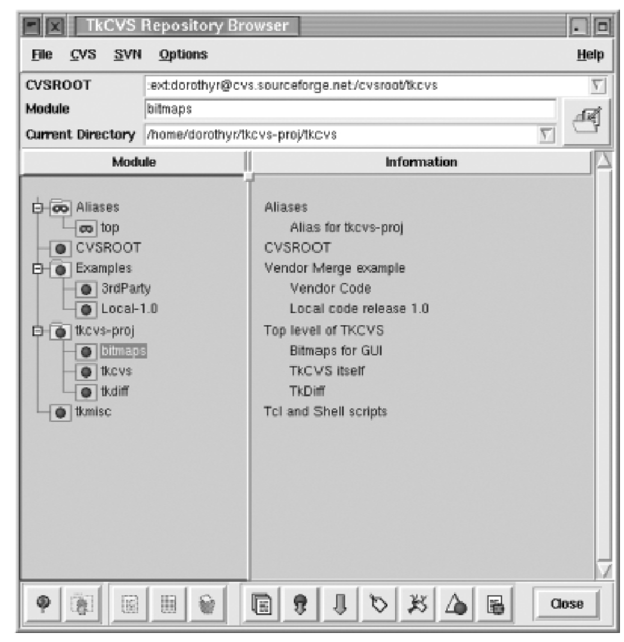


Figura 5. gCVS

En Debian deberemos hacer `apt-get install subversion`, si se desea publicar los repositorios en Apache2 `apt-get install Apache2-common` y el módulo específico `apt-get install libApache2`-subversion.

- Primer paso: Crear nuestro repositorio, usuario (consideramos que el usuario es svuser), grupo (svgroup) como *root*...

```
mkdir -p /usr/local/svn
addgroup svgroup
chown -R root.svgroup /usr/local/svn
chmod 2775 /usr/local/svn
```

- `addgroup svuser svggroup` Agrego el usuario svuser al grupo svgroup.

- Nos conectamos como svuser y verificamos que estamos en el grupo svgroup (con el comando group).

- `svnadmin create /usr/local/svn/pruebas`
Este comando creará un serie de archivos y directorios para hacer el control y gestión de las versiones. Si no se tiene permiso en /usr/local/svn se puede hacer en el directorio local: `mkdir -p $HOME/svndir` y a continuación `svnadmin create $HOME/svndir/pruebas`.

- A continuación creamos un directorio temporal `mkdir -p $HOME/svntmp/pruebas` nos pasamos al directorio cd $HOME/svntmp/pruebas y creo un archivo por ejemplo: `echo Primer Archivo Svn `date` > file1.txt`.

- Lo trasladamos al repositorio: Dentro del directorio hago `svn import file:///home/svuser/svndir/pruebas -m "Ver. Inicial"` Si lo hemos creado en `/usr/local/svn/pruebas` deberíamos poner el path completo después de file://. El import copia el árbol de directorios y el -m permite indicarle el mensaje de versión. Si no ponemos -m se abrirá un editor para hacerlo (se debe poner un mensaje para evitar problemas). El subdirectorio $HOME/svntmp/pruebas es una copia del trabajo en repositorio y es recomendable borrarla para no tener la tentación o el error de trabajar con ella y no con el repositorio (`rm -rf $HOME/svntmp/pruebas`).

- Una vez en el repositorio se puede obtener la copia local donde podremos trabajar y luego subir las copias al repositorio, para lo que hacemos:

```
mkdir $HOME/svm-work
cd $HOME/svn-work
svn checkout file:///home/svuser/svndir/pruebas
```

  Donde veremos que tenemos el directorio pruebas. Se puede copiar con otro nombre agregando al final el nombre que queremos. Para añadirle un nuevo fichero:

```
cd /home/kikov/svn-work/pruebas
echo Segundo Archivo Svn `date` > file2.txt
svn add file2.txt
svn commit -m"Nuevo archivo"
```

  Es importante remarcar que una vez en la copia local (svn-work) no se debe indicar el path. svn add marca para añadir el fichero al repositorio y realmente se añade cuando hacemos un cvn commit. Nos dará algunos mensajes indicándonos que es la segunda versión.

Si agregamos otra línea la file1.txt con echo 'date'>>file1.txt, luego podemos subir los cambios con: `svn commit -m"Nueva línea"`.

Es posible comparar el archivo local con el archivo del repositorio, por ejemplo, agregamos una tercera línea a file1.txt con echo 'date'>>file1.txt, pero no lo subimos y si queremos ver las diferencias, podemos hacer: svn diff.

Este comando nos marcará cuáles son las diferencias entre el archivo local y los del repositorio. Si lo cargamos con `svn commit -m"Nueva línea2"` (que generará otra versión) luego el `svn diff` no nos dará diferencias.

También se puede utilizar el comando `svn update` dentro del directorio para actualizar la copia local. Si hay dos o más usuarios trabajando al mismo tiempo y cada uno ha hecho una copia local del repositorio y la modifica (haciendo un commit), cuando el segundo usuario vaya a hacer el commit de su copia con sus modificaciones, le dará un error de conflicto, ya que la copia en el repositorio es posterior a la copia original de este usuario (es decir, ha habido cambios por el medio), con lo cual, si el segundo usuario hace el commit, perderíamos las modificaciones del primero. Para ello, debemos hacer un `svn update` que nos indicará el archivo que crea conflicto con un C y nos indicará los archivos donde se han puesto las partes en conflicto. El usuario deberá decidir con qué versión se queda y si podrá hacer un commit.

Un comando interesante es el `svn log file1.txt`, que nos mostrará todos los cambios que ha habido en el fichero y sus correspondientes versiones.

Un aspecto interesante es que el subversion puede funcionar juntamente con Apache2 (y también sobre SSL) para acceder desde otra máquina (consultar los clientes en http://svnbook.red-bean.com/) o simplemente mirar el repositorio. En Debian Administration explican cómo configurar Apache2 y SSL para Sarge, o como ya indicamos en la parte de servidores. Para ello, es necesario activar los módulos de WebDAV (ver http://www.debian-administration.org/articles/285 o en su defecto http://www.debian-administration.org/articles/208.

Como *root* hacemos:

```
mkdir /subversión chmod www-data:www-data
```

Para que Apache pueda acceder al directorio

```
svnadmin create /subversión
```

creamos el repositorio

```
ls -s /subversion
-rw-r--r-- 1 www-data www-data 376 May 11 20:27 README.txt
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 conf
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 dav
```

```
drwxr-xr-x 2 www-data www-data 4096 May 11 20:28 db
-rw-r--r-- 1 www-data www-data 2 May 11 20:27 format
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 hooks
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 locks
```

Para autentificación utilizamos htpasswd (por ejemplo con `htpasswd2 -c -m /subversion/.dav_svn.passwd` user creado como www-data. La -c sólo hay que ponerla la primera vez que ejecutamos el comando para crear el archivo. Esto indica que para acceder a este directorio se necesita passwd (que es el que hemos entrado para user).

Luego se debe cambiar el httpd.conf para que sea algo como:

```
<location /svn>
  DAV svn
  SVNPath /subversion
  AuthType Basic
  AuthName "Subversion Repository"
  AuthUserFile /subversion/.dav_svn.passwd
  Require valid-user
</location>
```

Reiniciamos Apache y ya estamos listos para importar algunos archivos, por ejemplo:

```
svn import file1.txt http://url-servidor.org/svn \
-m "Import Inicial"
```

Nos pedirá la autentificación (user/passwd) y nos dirá que el fichero file1.txt ha sido añadido al repositorio.

Actividades

1) Definir en PostgreSQL una DB que tenga al menos 3 tablas con 5 columnas (de las cuales 3 deben ser numéricas) en cada tabla.

Generar un listado ordenado por cada tabla/columna. Generar un listado ordenado por el mayor valor de la columna *X* de todas las tablas. Cambiar el valor numérico de la columna *Y* con el valor numérico de la columna *Z* + valor de la columna *W*/2.

2) El mismo ejercicio anterior, pero con MySQL.

3) Configurar el CVS para hacer tres revisiones de un directorio donde hay 4 archivos .c y un makefile. Hacer una ramificación (*branch*) de un archivo y luego mezclarlo con el principal.

4) Simular la utilización de un archivo concurrente con dos terminales de Linux e indicar la secuencia de pasos para hacer que dos modificaciones alternas de cada usuario queden reflejadas sobre el repositorio CVS.

5) El mismo ejercicio anterior, pero uno de los usuarios debe conectarse desde otra máquina al repositorio.

6) Ídem 3, 4 y 5 en Subversion.

Otras fuentes de referencia e información

[Debc, Ibi, Mou01]

PgAccess: http://www.pgaccess.org/

WebMin: http://www.webmin.com/

Mysql Administrator
http://www.mysql.com/products/tools/administrator/

Interfaces gráficas para CVS: http://www.twobarleycorns.net/tkcvs.html

O en la wiki de CVS:
http://ximbiot.com/cvs/wiki/index.php?title=CVS_Clients

Subversion: http://subversion.tigris.org

Free Book sobre Subversion: http://svnbook.red-bean.com/index.es.html

Apache2 y SSL: http://www.debian-administration.org/articles/349

Apache2 y WebDav: http://www.debian-administration.org/articles/285

Existe gran cantidad de documentación sobre Apache y SSL + Subversion en Debian además de http://www.debian-administration.org, poner en Google “Apache2 SSL and Subversion in Debian” para obtener algunos documentos interesantes.